Reference Guide to Educational Software minpy

Nadia Udler

nadiakap@optonline.net

This educational software helps in understand principles of learning algorithms in machine learning and artificial intelligence.

Abstract

Optimization lies at the heart of machine learning. Machine learning models optimize the loss function in order to fit the training data. The next step – even more challenging – is optimization of hyperparameters of the model. Python library `minpy` allows to create customized derivative free learning algorithms with desired properties[1] by combining building blocks from this library or other software. The library is intended primarily for educational purposes and its focus is on transparency of the methods rather than on efficiency of implementation. It allows to create customized iterative algorithm for training machine learning models. The user can control each element of the algorithm, such as starting point(s), direction of move at each iteration, step size at each iteration, and the stopping criterion.
Suggested exercises and student programming projects are provided.

Keywords: Learning Algorithm, Optimization Algorithm, Local and Global optimization, Derivative free optimization

---

[1] Properties of optimization algorithm include ability to search more locally or globally depending on the landscape of objective function, ability to adjust to different landscape, ability to deal with non smoothness.

Reference Guide to Educational Software minpy

Software Description

Python library `minpy` provides building blocks for the algorithm (see Appendix) that is based on the systematic approach to constructing optimization methods that use potential theory, described in [1,2]. These methods use only objective function values to find the search direction. Derivatives of objective function are not used. Convergence of such algorithms is proved in [1]. Three working algorithms for function minimization are implemented, NM, NMext, NMimproved. Five test functions are included, Booth, Sphere, Ackley, Easom, Shor. Their description can be found in [3]. The main class that contains building blocks for the algorithms is called `minimization`. Building blocks for algorithms and complete algorithms are implemented as methods of this class. There is helping class `Trial` that allows to keep information about newly computed direction of move before it is being tested for acceptance. The most general operation for finding direction of move at each iteration is *space dilation operator* shown in Fig.1. This transformation appears in Shor-r algorithm for minimization (see [4]) as heuristic rule. In [1] and [2] space dilation operator is derived from the condition of finding the best correcting transformation.

The following terms are used in the algorithm descriptions:

point - a vector in n-dimensional space in which function is defined,
polytope - geometric object with "flat" sides, for example, triangle and square are polytopes,
vertices of polytope are *points*.
Iteration of the algorithm is described in terms of a movement of vertices of a polytope,
"best" point– point with smallest function value among all vertices of the polytope,
"worst" point– a point with largest function value among all vertices of the polytope,
simplex - a special type of polytope of n vertices in (n+1) dimensions,
center of mass – center of mass of a polytope,
reflection – reflection of a vertex of a polytope about center of mass of other vertices
space dilation operator – matrix $R_\beta(\theta) = I + (1-\beta)\theta\theta^T$ that is applied to a *point X.*

Class `minimization`

This is `minimization` description.

`minpy.minimization`(*f, x0, dist="uniform", K=4, line_search_method=None,step_size = 0.5, tol=0.0001, maxiter=200,*)

Parameters
f **-** The objective function to be minimized.
x0 - Initial guess.
dist **–** initial distribution of trial points.
K – number of points in the initial trial,
line_search_method **–** line search method used to find optimal step size in the given direction.

step_size – step size in the given direction
tol – absolute error in function values that is acceptable for convergence
maxiter – maximum number of algorithm iterations allowed

Methods.

`minpy.minimize.initialize`**()**

Creates initial population of points according to desired distribution. Distribution must be set through parameter ***dist*** at the creation of class `minimization`.

`minpy.minimize.get_f`**(x)**

Compute function values at x

`minpy.minimize.sort`**()**

Sort population of points according to function values

`minpy.minimize.reflect`**(X)**

Returns reflection of X about center of mass of current population of points.

`minpy.minimize.expand`**(X)**

Returns expanded reflection of X about center of mass of current population of points.

`minpy.minimize.contract`**(X,s)**

Returns contracted reflection of X about center of mass of current population of points.

`minpy.minimize.modify`**(X,s)**

Returns reflected, expanded, or contracted reflection of X about center of mass of current population of points, where s>1 corresponds to reflection, s<-1 corresponds to expansion, -1<X<0 corresponds to contraction.

`minpy.minimize.compute_new_vertex`**(X,s)**

Computes direction and movement length for the next iteration according to condition of local improvement given in [1]. Uses X and center of mass of all points except "worst" to find direction of the move.

`minpy.minimize.space_dilation`**(beta, theta)**

Computes space dilation matrix $R_\beta(\theta) = I + (1-\beta)\theta\theta^T$.

`minpy.minimize.space_transformation`**(X, beta, theta)**

Applies space dilation matrix $R_\beta(\theta) = I + (1-\beta)\theta\theta^T$ to point X. Figure 1 demonstrates the application of space dilation operator to point X for different values of parameter β.

`minpy.minimize.space_dilation_inv`**(X, beta, theta)**

Computes inverse of space dilation matrix $R_\beta(\theta)$. Inverse of $R_\beta(\theta) = I + (1-\beta)\theta\theta^T$ is given by $R_{\beta inv}(\theta) = I + (1-1/\beta)\theta\theta^T$

`minpy.minimize.householder_reflection`(X, w)

Applies Householder matrix (variant of space dilation matrix when $\beta = -1$) $H = I - 2ww^T$ to point X. Figure 2 demonstrates Householder reflection.

`minpy.minimize.NM`()

Computes minimum of objective function f using Nelder and Mead algorithm. This method uses `reflect`, `expand` and `contract` methods.

`minpy.minimize.NMext`()

Computes minimum of objective function f using Nelder and Mead algorithm. This method uses `modify` method.

`minpy.minimize.NMImp`(*X,s*)

Improves current value of objective function by making several steps toward "better" objective. On each step `compute_new_vertex` is called.

`minpy.minimize.create_trial`(*X*)

Creates trial structure that holds argument vector X, function values at X, and proposed center of mass of X except "worst".

`minpy.stop`()

Stopping criterion.

`minpy.kl_difergence`(X,X_pred)

returns Kullbeck-Leibler divergence between true distribution X and predicted distribution X_pred

Examples

Below is an example of calling `NM` method to find minimum of Sphere function of 3 arguments:

```
import numpy as np
import minpy as mp
X0 = np.array([2.0, 1.0, 1.9])
sphere = mp.Minimization(mp.spher,X0)
res = sphere.minimizeNM()
print('function value:',res[0])
print('argument values at optimal point:',res[1])
```

Class `trial`

This is `trial` description.

`minpy.trial`(*u, fu, m*)

Parameters

u – population of points

fu – population of function values at u

m – center of mass of all points except “worst”. Point

Methods.

This class does not have methods

Examples

Below is an example of `trial` usage:

```
X=numpy.array(0.5,0.5)
trial = self.create_trial(X)
if trial.fu < current_fu:
    self.u = trial.u
    self.fu = trial.fu
```

Class `test_functions`

This is `test_functions` description. This class contains four test functions.

Methods.

`minpy.Easom`**(x)**

Easom function

`minpy.Sphere`**(x)**

Sphere function

`minpy.Booth`**(x)**

Booth function

`minpy.Ackley`**(x)**

Askley function

Installation requirements

Current version of `minpy` is built with Python 3.6. and `numpy` 1.18.2.

Questions and Exercises for Students

1. How do you specify the number of function arguments when optimizing it using `minpy`.
2. How do you specify the size of initial population when using minimization functions from `minpy`.
3. How do you specify initial distribution of points when using minimization functions from `minpy`.

Programming Projects for Students

1. Create an algorithm based on minpy building blocks that performs iteration steps according to (28), where initial distribution of points follows normal distribution centered at initial point with unit variance. Keep population size at each iteration unchanged. Use stopping criterion of your choice. Test this algorithm on Ackley function.

2. Create an algorithm based on minpy building blocks that performs iteration steps according to (28), where initial distribution of points follows exponential distribution with unit variance. Use stopping criterion of your choice. Test this algorithm on Sphere function with 5 arguments.

3. Create an algorithm based on minpy building blocks that performs iteration steps according to (28), where initial distribution of points follows normal distribution centered at initial point with unit variance, and on each iteration 3 random points with "good" values (values that are smaller than average function value achieved so far) are added to the population. The algorithm should stop when largest distance between points becomes less than certain small number. Test this algorithm on Booth function.

4. Create function that computes the cost of the travel for traveling salesman problem. Use this function to test NMExt algorithm from minpy.

5. Create function that computes the weights of items in the knapsack in the knapsack problem. Use this function to test NMExt algorithm from minpy.

6. Create an algorithm with a "memory". On each iteration of the algorithm the new direction is computed as a difference between two previous directions. Algorithm starts from randomly selected simplex with number of points n+1, where n is number of function arguments. Second steps of the algorithm should be computed as reflection of worst initial point from the center of mass of "good" points. Use stopping criterion of your choice. Test on Sphere function in 3 dim and 5dim space.

7. Implement a variant of Shot-r algorithm. The steps of the algorithm are listed below. Use stopping criterion of your choice. Test on Ackley function.

8. Implement a variant of CMAES. The algorithm should stop when Kullback – Leibler difergence becomes smaller than some given tolerance.

References

[1]Kaplinsky, A. I., Propoi A.I. (1994). First-order nonlocal optimization methods that use potential theory, *Automation and Remote Control*.

[2]Philip E Gill, Walter Murray, Margaret H. Wright, 1982. *Practical Optimization,* Emerald Group Publishing.

[3]Test functions for optimization, Wikipedia
https://en.wikipedia.org/wiki/Test_functions_for_optimization

[4]N. Z. Shor, Minimization Methods for non-differentiable functions, Springer, 1985

Footnotes

[1]*properties of optimization algorithm* include ability to search more locally or globally depending on the landscape of objective function, ability to adjust to different landscape, ability to deal with non smoothness.